Learning Python

Getting results for beamlines and scientific programming

3. Basic Python: assignments and expressions

U.S. DEPARTMENT OF **ENERGY** | Office of Science

Outline of topics to be covered

1. Assignment statements
   - Assignment to a tuple
   - Equals for complex data types
   - Unassignment
2. Expressions
3. Comparisons
4. Misc. rules for coding: Comments & continuations

5/10/11   2

Assignment

- A value is associated with a variable using the "=" sign
  - A = "this is a string"
  - A = 1.1
  - A = 1

  It is better to think of the = operation as assigning a pointer to a value and placing that pointer into a variable

- Python allows operators to be combined with assignment, called augmented assignment, for example:

  A += 1
  - This adds 1 to A (produces an error if A is not defined)

  other commonly used augmented assignment combinations include:
  - *=, -=, |= (Or), &= (And)

5/10/11   3

Assignment to a tuple of variables

- Equals can be used to "break up" a tuple or list

```
>>> (a,b,c) = (2,4,(6,8))
>>> a
2
>>> b
4
>>> c
(6, 8)
```
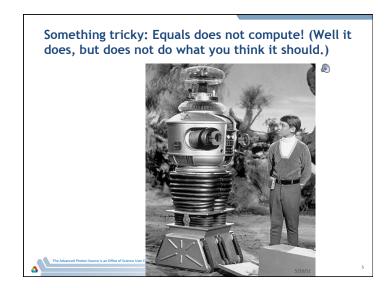
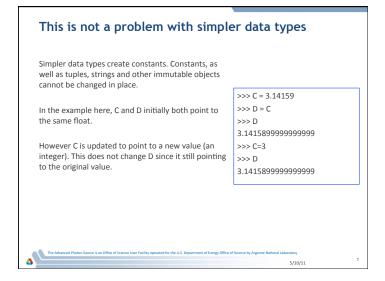- Note that frequently this is coded without the parenthesis

```
>>> a,b,c = [2,4,(6,8)]
```

- The lengths of the tuples, etc. must match

```
>>> a,b,c,d = [2,4,(6,8)]
ValueError: need more than 3 values to unpack
```

5/10/11   4

Something tricky: Equals does not compute! (Well it does, but does not do what you think it should.)

The Advanced Photon Source is an Office of Science User F...

5/10/11   5

Equals creates a pointer to an object but not a new copy of the object

This means that when you use equals you are just adding a new label to the same object

- Here we create an object (a list) that is pointed to by variable A and B

```
>>> cvar = complex(0,1)
>>> A = [1, 3.14159, 'pi', ['a','list'], cvar]
>>> B = A
```

- id() shows us the memory location for each variable. Note they are the same.

```
>>> id(A)
3657936
>>> id(B)
3657936
```

- If we make a change to A, then B changes as well, since they both refer to the same list

```
>>> B[1]
3.1415899999999999
>>> A[1] = 3.0
>>> B[1]
3.0
```

- It is possible to make a copy of an object. A slice creates a a new copy:

```
B = A[:]
```

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

5/10/11   6

This is not a problem with simpler data types

Simpler data types create constants. Constants, as well as tuples, strings and other immutable objects cannot be changed in place.

In the example here, C and D initially both point to the same float.

However C is updated to point to a new value (an integer). This does not change D since it still pointing to the original value.

```
>>> C = 3.14159
>>> D = C
>>> D
3.1415899999999999
>>> C=3
>>> D
3.1415899999999999
```

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

5/10/11   7

The same thing happens with compound objects

- Here we create c as a tuple containing a tuple (a) and a list (b)
  - Thus the tuple contains pointers to two objects

- The list can be changed, as b[1] or c[1][1]. If we change either, both b & c change.

- Note that assigning a new value to b does not change c

```
>>> a = (1,2)
>>> b = [3,4]
>>> c = (a,b)
>>> c
((1, 2), [3, 4])
>>> b[1]=99
>>> c
((1, 2), [3, 99])
>>> c[1][1] = -1
>>> b
'test'
>>> c
((1, 2), [3, -1])

>>> b = 'test'
>>> c
((1, 2), [3, -1])
```

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

5/10/11   8

Python Course Handouts - Brian Toby

Be warned!

This process by which equals creates pointers to objects is one of the most confusing parts of Python.

- There are good reason why Python copies pointers to objects rather than contents:
  - Saves space
  - Faster
  - Sometimes very convenient

- However, this is also one of the major sources for errors in Python code!

If you find that objects are changing their values unexpectedly, it is probably because they are referenced in more than one place – be careful when nesting complex data items and consider using copy, indexing or slicing during assignments.

5/10/11   9

Unassignment

- Usually one allows Python to clean up unused variables automatically when they are no-longer needed. When we discuss functions, we will see that local variables are created as needed and when they are no longer needed, the memory is reclaimed.

- However, if one wants to remove a variable, the del statement can be used:

  A = [1,2,3]
  del A

- Note that del is better thought of as "forget" than "delete" as it will not necessarily reclaim space

  A = [1,2,3]
  B = A
  del A

  - In this case the del statement does not release the memory associated with the list because B still refers to it. However, if B is also deleted and not referenced elsewhere, Python will then release the memory for reuse.

5/10/11   10

Expressions

- Expressions are formed using operators (+, **) or functions [pow()] and constants or variables:
  - A = 2 + 2
  - B = var1 * var2
  - { 2: 2**10, 3: pow(3,10), }
  - 3*"Spam "
- Expressions can be used just about anywhere a value of the appropriate type is used

```
>>> { 2: 2**10, 3: pow(3,10), }
{2: 1024, 3: 59049}
>>> print 3*'Spam '
Spam Spam Spam
>>>
```

- You will see a lot more expressions as we learn more Python functionality, but see http://software-carpentry.org/4_0/python/basics/ if you are impatient.

5/10/11   11

Comparisons

Comparisons are logical expressions that have a result of True or False. (Note capitalization). Comparisons are usually used for tests.

```
>>> true
NameError: name 'true' is not defined
>>> True
True
>>> 2 >= 4
False
>>> 2 < 4
True
```

- In addition to the expected comparisons, <, <=, >, >=, ==, !=, there are some special ones: in, not in, is, is not
  - is/is not tests object identity (to be explained later)
  - in/not in tests to see if an item is matched in a list (etc.)

```
2 in [2,3,4]        is True
2 in [(2,3),4]      is False
(2,3) in [(2,3),4]  is True
2,3 in [(2,3),4]    is (2,False)
```

Why? Because the last line is interpreted as
(2, (3 in [(2,3),4]))
and **3 in [(2,3),4]**
gets evaluated as **False**
That gives **(2, False)**

5/10/11   12

Python Course Handouts - Brian Toby

Combining comparisons

Python has Boolean operators: **not**, **and**, **or** than can be used to combine comparisons.

▪ Advice: Use parentheses liberally to make sure you are testing what you want. They don't cost anything.

```
A > B and C or A !=D
```
is the same as
```
(A > B and C) or (A !=D)
```
but is not the same as
```
(A > B) and (C or A !=D)
```

5/10/11   13

Commenting code

There are two ways to insert comments into code

▪ Anything beginning with a hash or pound character (#) is ignored, except inside a string
```
# this is a comment
if A == True:      # N.B. not same as if A: for non-boolean
  pass
S = "this string includes # hash"  # comment
```
– Note the 2nd hash (#) starts a comment but not the 1st

▪ You can define character string on one or more lines that is not assigned to any variable – called a docstring
```
'this is a comment'
'''this is a
Multiline comment'''
```
– docstrings are preferred to document routines and classes (see http://www.python.org/dev/peps/pep-0257/)

5/10/11   14

Continuing lines

One can break up a command between grouping characters (), {}, [] with no extra actions. The Python interpreter knows the next line is part of the previous command because the command is otherwise incomplete.
```
A = (
 1,        2, 3,
 )
A = [
 1,
 2,
 3,
 ]
A = {      1:1, 2:2,
 3:'should not happen'}
```
▪ It is also possible to use backslashes to continue lines, but don't do this, since one can always use parenthesis

5/10/11   15

Homework

▪ What is the result of:
```
1 + 2 * 3
```
▪ What will the value of B be after executing these three lines of code:
```
A = [2,3,4]
B = (A,A)
A[2]=5
```
▪ Rewrite this comparison in a simpler form:
```
a < 4 or a > 4
```

▪ Which of these code continuations are valid?

```
A = [1,2,3
          ]
```
```
A = 1,2,
     3
```
```
A = (1,2,
     3)
```
```
A = (
        1
        +
        1
     )
```

5/10/11   16

Python Course Handouts - Brian Toby